LabVIEW Programming for a Multicore Environment

Stefan Kreuzer
Applications Engineer
National Instruments
Agenda

• Overview of LabVIEW Multithreading
• Parallel Programming Techniques
• Real-Time Considerations
• Resources
Impact on Engineers and Scientists

Engineering and scientific applications are typically on dedicated systems (i.e. little multitasking).
Creating Multithreaded Applications

Engineers and scientists **must** use threads to benefit from multicore processors.
Automatic Multithreading in LabVIEW

- LabVIEW automatically divides each application into multiple execution threads
- LabVIEW introduced multithreading in 1998
Automatic Multithreading in LabVIEW

- LabVIEW automatically divides each application into multiple execution threads
- LabVIEW introduced multithreading in 1998
Demo:
“Accidental Parallelism” in LabVIEW
NEW Multithreading Features in LabVIEW 8.5

• Scale # of execution system threads based on available cores
• Improved thread scheduling for LabVIEW timed loops
• Set Processor Affinity with Timed Structures
• Real-Time Features:
  – Support for Real-Time targets with Symmetric Multiprocessing
  – Real-Time Execution Trace Toolkit 2.0
Eaton created a portable in-vehicle test system for truck transmissions using LabVIEW.

- Acquired and analyzed 16 channels on single core using DAQmx multithreaded driver
- Now acquire and analyze 80+ channels on multicore

“There was no need to rewrite our application for the new multicore processing platforms.”

Scott Sirrine
Lead Design Engineer
Eaton Truck Division
## Multithreaded Software Stack Support

<table>
<thead>
<tr>
<th>Software Stack</th>
<th></th>
<th>Example:</th>
<th>Multithreaded nature of LabVIEW and structures that provide optimization</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Development tool</strong></td>
<td>Support provided on the operating system of choice; tool facilitates correct threading and optimization</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td><strong>Libraries</strong></td>
<td>Thread-safe, re-entrant libraries</td>
<td>✓</td>
<td>BLAS libraries</td>
</tr>
<tr>
<td><strong>Device drivers</strong></td>
<td>Drivers designed for optimal multithreaded performance</td>
<td>✓</td>
<td>NI-DAQmx driver software</td>
</tr>
<tr>
<td><strong>Operating system</strong></td>
<td>Operating system supports multithreading and multitasking and can load balance tasks</td>
<td>✓</td>
<td>Support for Windows, Mac OS, Linux® OS, and real-time operating systems</td>
</tr>
</tbody>
</table>
Evaluating if existing LabVIEW applications will run faster on multicore systems

Key Considerations

• Is the code sequential or parallel?
• How are shared resources being utilized within the application?

Conclusions

• Code speed-up will widely vary depending on application
• Many applications will require minor modifications to fully optimize for multicore
Sequential vs. Parallel Code

**Sequential Example**
- LabVIEW will execute VIs in order, one after another.

**RESULT:**
A multicore system will not improve processing speed over a single core system, because there is no parallelism in the code.
Sequential vs. Parallel Code

Parallel Example
- Since code has parallel branches, LabVIEW will try to compile code for parallel execution.

RESULT:
Code can now take advantage of multicore systems, i.e. “Filter” and “Spectral Measurements” VIs will execute in parallel.
Potential Bottlenecks: Shared Resources

1. Data Dependencies
   
   **Example:** Data stored in global variables that need to be accessed by different VIs would be a shared resource

2. Hard Disk
   
   **Example:** Computers can only read or write to the hard disk one item at a time (File I/O cannot be made into parallel operations)

3. Non-reentrant VIs
Non-reentrant VIs

• VIs that are non-reentrant cannot be called simultaneously, one call runs and the other call waits for the first to finish before running.

To make a VI reentrant, select **File»VI Properties**, select the **Execution** category and choose **Reentrant execution**.
Parallel Programming Techniques to Improve Performance on Multicore Systems

- Task Parallelism
- Data Parallelism
- Pipelining
Data Parallelism

You can speed up processor-intensive operations on large data sets on multicore systems.
Data Parallelism

You can speed up processor-intensive operations on large data sets on multicore systems.
Data Parallelism Demo
Application Example: High-Speed Control

- Max Planck Institute (Munich, Germany)
- Plasma control in nuclear fusion tokamak with LabVIEW on 8 core system using data parallelism technique

“...with LabVIEW, we obtained a 20X processing speed-up on an octal-core processor machine over a single-core processor...”

Louis Giannone
Lead Project Researcher
Pipelining Strategy

- Many applications involve sequential, multistep algorithms
- Applying pipelining can increase performance
Pipelining Strategy

- Acquire
- Filter
- Analyze
- Log

Time progression: $t_0$, $t_1$, $t_2$, $t_3$
Pipelining Strategy

- Acquire
- Filter
- Analyze
- Log

CPU Core

$t_0$, $t_1$, $t_2$, $t_3$
Pipelining Strategy

- CPU Core
  - Acquire
  - Filter
  - Analyze
  - Log
- CPU Core
  - Acquire
  - Filter
  - Analyze
  - Log
- CPU Core
  - Acquire
  - Filter
  - Analyze
  - Log
- CPU Core
  - Acquire
  - Filter
  - Analyze
  - Log

Time:
- $t_0$
- $t_1$
- $t_2$
- $t_3$
Pipelining in LabVIEW

Sequential

Pipelined

Note: Queues may also be used to pipeline data between different loops
Pipelining Demo
Key Considerations for Pipelining

- Consider number of Processor Cores when determining number of pipeline stages
- Be sure to balance stages since longest stage will be limiting factor in performance speed-up
  - Example: Unbalanced

Diagram:
- Non-Pipelined (total time = 4s)
  - Stage 1 (3s)
  - Stage 2 (1s)
- Pipelined (total time = 3s): Speed-up = 1.33X (not an ideal case for pipelining)
  - Stage 1 (3s)
  - Stage 2 (1s)
Tips for Balancing Pipeline Stages

• Use LabVIEW benchmarking techniques
  – Perform basic benchmarking with timestamps and VI Profiler
Deterministic Real-Time Systems

LabVIEW 8.5 adds Symmetric Multiprocessing (SMP) for real-time systems.
Assigning Tasks to Specific Cores

In LabVIEW 8.5, users can assign code to specific processor cores using the LabVIEW Timed Loop.
Processor Affinity with the Timed Loop

- Recommended for Real-Time development
- Under special use-cases it can be used on Windows (example: cache optimization)
Execution Trace Toolkit 2.0

- 30-day evaluation
- Compatible with LabVIEW 7.1 and higher
- **New Features in Version 2.0**
  - Performance improvements
  - Trace activity sorting
  - New debugging flags (priority inheritance)
  - Multicore support
Application Example: Real-Time Control

- Wind Tunnel Safety-of-Flight system at NASA Ames Research Center

- Benchmarks Results
  - Ran on PXI-8106 RT Controller
  - Time-Critical loop was reduced from **43 % CPU load** to **30 % CPU load** on one CPU of the PXI-8106RT
  - A core leftover for processing of non-critical tasks.

Image Source: http://windtunnels.arc.nasa.gov
Resources
www.ni.com/multicore